Intelligent Video Surveillance Systems:
Data Preprocessing & Data Loader, **Start:** October 13, 2021
1st Homework Assignment. **Deadline:** 23:59 October 29, 2021

Introduction

Data preprocessing and data loading are the crucial steps before a learning stage in the most of tasks. We have to make sure that our data samples are correctly loaded and transferred from a disk into a memory. In the surveillance systems, we mostly consider images as an input with corresponding ground truths (i.e., labels) in the terms of supervised learning.

Thus, in this homework assignment, you will implement a class, regarding to OOP (Object Oriented Programming) principles, in Python language with a PyTorch module. Your class must inherit appropriate class from the PyTorch module and override a few methods to properly implement a custom dataset compatible with the PyTorch data loader and algorithms. The class will also contain the image preprocessing and data augmentation techniques.

The project includes 5 RGB images with the image segmentation labels, so you will implement methods to load these images with labels in the prepared `CustomDataset` class in `CustomDataloader.py` source file. The image semantic segmentation task will be precisely presented later on this course; however, you should be able to sufficiently analyse and understand the structure of semantic segmentation labels [1].

To successfully fulfil this assignment, please, carefully follow the objectives!

Objectives

Overall description:

- All of the methods will be implemented in the prepared Python class `CustomDataset` in `CustomDataloader.py` source file,
- briefly comment every implemented method/function,
- write down a documentation, concisely but clearly describe: title page, problem analysis, your solution, results (visualisations) and conclusion.

Dataset structure:

- The **dataset** directory contains two sub-folders – **images** and **labels**. The pair of training sample (image and label) shares exactly the same name and the image format,
- each label is stored in train ID fashion (see the explanation below),
- each image consists of 3-channels with 8-bit unsigned integers (RGB image) in PNG format,
- there are 19 different classes + 1 extra for unlabelled pixels: any index from 0 to 18 belongs to classes, index 255 belongs to unlabelled pixel (see the `labels.py` source code).

* Train ID fashion means that the label consists of only 1-channel (i.e., grayscale image) with 8-bit unsigned integers. In the other words, each pixel contains only single number with respects to the training classes. For instance, a training set that includes cars, buses, roads and pedestrians, has 4 classes in the total. Subsequently, the pixels in the labels can only take the following values: 0, 1, 2 or 3.

[1] https://learnopencv.com/pytorch-for-beginners-semantic-segmentation-using-torchvision/

Dataloader implementation:

- Inherit the proper class and override (implement) the methods for data loading based on PyTorch tutorial [2]. In your solution, you will not load data from a **csv** file, but directly from the **images** and **labels** directories! To iterate through the directories, you can use `os` and `os.path` modules. Python also provides useful methods for `string`, such as `replace`, which could be useful in this task. To load raw image, we highly recommend to make use of `Pillow` module [3] (`PIL` image). You can utilise `numpy` module that allows you to easily convert image from `PIL` to `numpy array`, `tensor` and vice versa,

- implement or use random `horizontal flip` and `crop image` augmentation techniques with 100% of the probability rate. Set the crop size to 512×512 pixels. If you directly use torchvision implementations, be sure that these transformations are applied for image as well as for label (the implementations are stochastic!),

- normalise the image after augmentation with $\mu = [0.485, 0.456, 0.406]$ and $\sigma = [0.229, 0.224, 0.225]$,

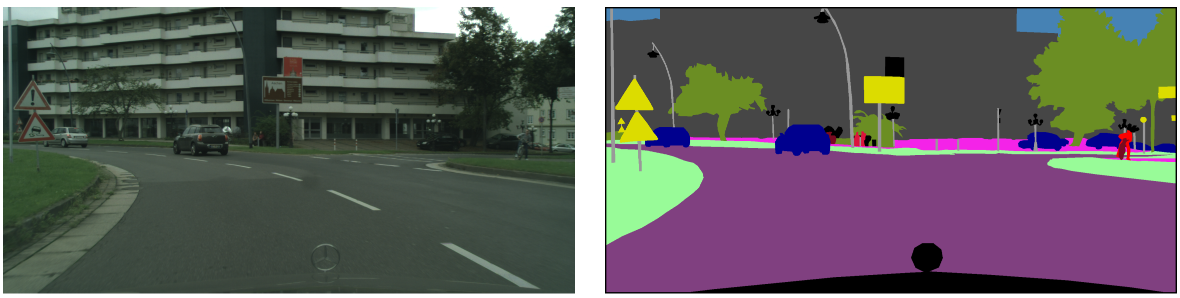
- use the mentioned augmentation techniques and the normalisation within your overridden method in the correct order, and return 2 instances of the `tensor` class – transformed **image** and **label**,

- implement `shows` method to visualise loaded image with **colourful** label. An example of colourful label is depicted in the Figure 1. You can use `matplotlib` module. To convert label into colourful image, we prepared pre-defined colours for each train ID value, see `labels.py` source code. Do not forget that input is `tensor` class with the **normalised** image, you have to perform image de-normalisation and convert it into correct image structure (class),

- complete the `main` function, where you will iterate throughout the whole dataset samples using `DataLoader` from PyTorch module (the input is your `CustomDataset` class). Use your implemented `shows` method, inside this loop, to visualise the images with the labels. Store each image and colourful label on the disk in PNG format to **output** directory. The saving process can be implemented inside the `shows` method for simplicity.

(***hint:** We do **NOT** recommend you to directly load image into a list/array but just the paths to the images with their labels! In the real experiments, you will face to **several thousand** or even **hundred thousand** of the training samples. Your memory is not unlimited!)

You will submit **two files** and **one directory**: your implementation of `CustomDataset` class in the `CustomDataloader.py` source file, **output** directory containing 5 images and 5 colourful labels (10 in the total) and the documentation in **PDF** format.



(a) A raw image. (b) A colourful label.

Figure 1: An example of the desired output.

[2] https://pytorch.org/tutorials/recipes/recipes/custom_dataset_transforms_loader.html
[3] https://pillow.readthedocs.io/en/stable/handbook/tutorial.html#using-the-image-class